



TREINAMENTO NODE.JS



Callback - Entendendo

- Entender callbacks em JavaScript é uma das primeiras coisas que o desenvolvedor iniciante em JavaScript deveria saber mas também é um dos conceitos mais difíceis de entender.
- A razão pela qual precisamos usar **callbacks** em JavaScript é por causa do conceito de assincronicidade e do comportamento não bloqueante do JavaScript.
- Vamos imaginar uma situação que devemos escrever uma classe Pessoa e essa Pessoa acorda e segue uma rotina matinal (um passo por vez e em ordem):
 - Acordar
 - Colocar calças
 - Colocar camisa
 - Colocar sapatos
 - Ir para escola



Callback - Entendendo

Forma síncrona

```
function Person() {  
  this.wakeUp = function () {  
    console.log("Acordar");  
  };  
  this.putOnPants = function () {  
    console.log("Colocar calças");  
  };  
  this.putOnShirt = function () {  
    console.log("Colocar camisa");  
  };  
  this.putOnShoes = function () {  
    console.log("Colocar sapatos");  
  };  
  this.goToSchool = function () {  
    console.log("Ir para escola");  
  };  
}  
  
var person = new Person();  
  
person.wakeUp(); // we want to do this first  
person.putOnPants(); // and then this  
person.putOnShirt(); // then this  
person.putOnShoes(); // then this  
person.goToSchool();
```



Callback - Entendendo

Forma síncrona

```
function Person() {
  this.wakeUp = function () {
    console.log("Acordar");
  };
  this.putOnPants = function () {
    console.log("Colocar calças");
  };
  this.putOnShirt = function () {
    console.log("Colocar camisa");
  };
  this.putOnShoes = function () {
    console.log("Colocar sapatos");
  };
  this.goToSchool = function () {
    console.log("Ir para escola");
  };
}

var person = new Person();

person.wakeUp();           // we want to do this first
person.putOnPants();       // and then this
person.putOnShirt();       // then this
person.putOnShoes();       // then this
person.goToSchool();
```

Forma assíncrona sem callback

```
function Person() {
  this.wakeUp = function() {
    // Simula uma operação assíncrona de 1 segundo
    setTimeout(() => { console.log("Acordar"); }, 1000);
  };
  this.putOnPants = function() {
    // Simula uma operação assíncrona de 2 segundos
    setTimeout(() => { console.log("Colocar calças"); }, 2000);
  };
  this.putOnShirt = function() {
    // Simula uma operação assíncrona de 1.5 segundos
    setTimeout(() => { console.log("Colocar camisa"); }, 1500);
  };
  this.putOnShoes = function() {
    // Simula uma operação assíncrona de 0.8 segundos
    setTimeout(() => { console.log("Colocar sapatos"); }, 800);
  };
  this.goToSchool = function() {
    // Simula uma operação assíncrona de 1.2 segundos
    setTimeout(() => { console.log("Ir para escola"); }, 1200);
  };
}

var person = new Person();

person.wakeUp();           // we want to do this first
person.putOnPants();       // and then this
person.putOnShirt();       // then this
person.putOnShoes();       // then this
person.goToSchool();
```




Callback - Entendendo

Forma assíncrona sem callback

```
function Person() {
  this.wakeUp = function() {
    // Simula uma operação assíncrona de 1 segundo
    setTimeout(() => { console.log("Acordar"); }, 1000);
  };
  this.putOnPants = function() {
    // Simula uma operação assíncrona de 2 segundos
    setTimeout(() => { console.log("Colocar calças"); }, 2000);
  };
  this.putOnShirt = function() {
    // Simula uma operação assíncrona de 1.5 segundos
    setTimeout(() => { console.log("Colocar camisa"); }, 1500);
  };
  this.putOnShoes = function() {
    // Simula uma operação assíncrona de 0.8 segundos
    setTimeout(() => { console.log("Colocar sapatos"); }, 800);
  };
  this.goToSchool = function() {
    // Simula uma operação assíncrona de 1.2 segundos
    setTimeout(() => { console.log("Ir para escola"); }, 1200);
  };
}

var person = new Person();

person.wakeUp();           // we want to do this first
person.putOnPants();       // and then this
person.putOnShirt();       // then this
person.putOnShoes();       // then this
person.goToSchool();
```

Forma assíncrona com callback

```
function Person() {
  this.wakeUp = function(callback) {
    // Simula uma operação assíncrona de 1 segundo
    setTimeout(() => { console.log("Acordar"); callback(); }, 1000);
  };
  this.putOnPants = function(callback) {
    // Simula uma operação assíncrona de 2 segundos
    setTimeout(() => { console.log("Colocar calças"); callback(); }, 2000);
  };
  this.putOnShirt = function(callback) {
    // Simula uma operação assíncrona de 1.5 segundos
    setTimeout(() => { console.log("Colocar camisa"); callback(); }, 1500);
  };
  this.putOnShoes = function(callback) {
    // Simula uma operação assíncrona de 0.8 segundos
    setTimeout(() => { console.log("Colocar sapatos"); callback(); }, 800);
  };
  this.goToSchool = function() {
    // Simula uma operação assíncrona de 1.2 segundos
    setTimeout(() => { console.log("Ir para escola"); }, 1200);
  };
}

var person = new Person();

person.wakeUp(function() {
  person.putOnPants(function() {
    person.putOnShirt(function() {
      person.putOnShoes(function() {
        person.goToSchool();
      });
    });
  });
});
```


Callback Hell

- Termo usado para descrever a situação quando muitos callbacks aninhados são usados em operações assíncronas. Isso pode tornar o código difícil de ler e manter.

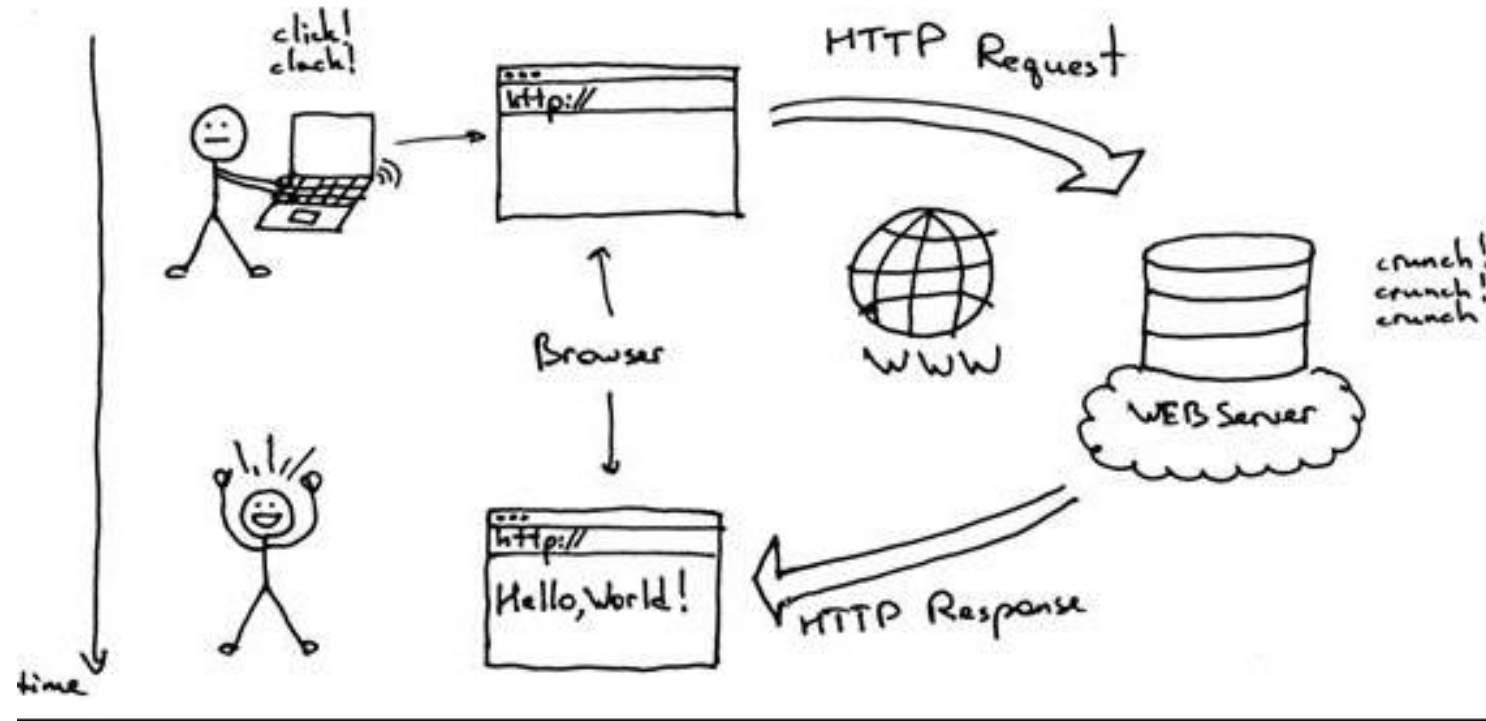
```

1  function hell(win) {
2    // for listener purpose
3    return function() {
4      loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5        loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6          loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7            loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8              loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9                loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                 loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                     async.eachSeries(SCRIPTS, function(src, callback) {
14                       loadScript(win, BASE_URL+src, callback);
15                     });
16                   });
17                 });
18               });
19             });
20           });
21         });
22       });
23     });
24   });
25 };
26 }

```



Web Server





Web Server

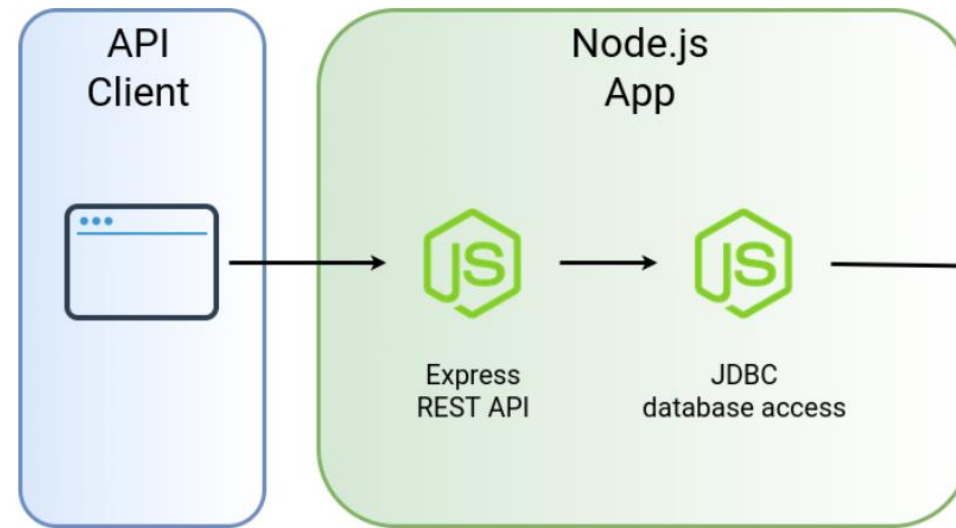
- Um **servidor web**, é um software responsável por atender solicitações HTTP (Hypertext Transfer Protocol) de clientes, como navegadores da web, e fornecer as respostas apropriadas, geralmente em forma de páginas da web, arquivos ou dados.
- Em Node.js, você pode criar um servidor web usando o módulo embutido chamado **http**.
- Este módulo fornece funcionalidades para criar servidores HTTP de forma muito simples.
- Ele não inclui um sistema de roteamento integrado. Para criar rotas personalizadas com o módulo **http**, você deve analisar a URL da solicitação manualmente e tomar decisões com base na rota solicitada.



Web Server - http

```
1 var http = require("http");
2
3 http.createServer(function (request, response) {
4     // Send the HTTP header
5     // HTTP Status: 200 : OK
6     // Content Type: text/plain
7     response.writeHead(200, {'Content-Type': 'text/plain'});
8
9     // Send the response body as "Hello World"
10    response.end('Hello World\n');
11 }) .listen(8081);
12
13 // Console will print the message
14 console.log('Server running at http://127.0.0.1:8081/');
```

Web Server - express



```
1  const express = require('express');
2  const app = express();
3
4  app.get('/ping', (request, response) => {
5    response.send('pong');
6  });
7
8  app.listen(8080, 'localhost');
```

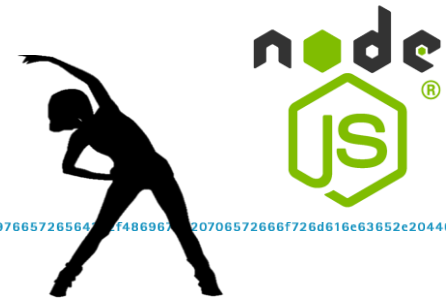
Método de Resposta

- Os métodos de resposta (res) podem enviar uma resposta ao cliente, e finalizar o ciclo solicitação-resposta.

<https://expressjs.com/pt-br/4x/api.html>

| Método | Descrição |
|-------------------------------|---|
| <code>res.download()</code> | Solicita que seja efetuado o download de um arquivo |
| <code>res.end()</code> | Termina o processo de resposta. |
| <code>res.json()</code> | Envia uma resposta JSON. |
| <code>res.jsonp()</code> | Envia uma resposta JSON com suporte ao JSONP. |
| <code>res.redirect()</code> | Redireciona uma solicitação. |
| <code>res.render()</code> | Renderiza um modelo de visualização. |
| <code>res.send()</code> | Envia uma resposta de vários tipos. |
| <code>res.sendFile</code> | Envia um arquivo como um fluxo de octeto. |
| <code>res.sendStatus()</code> | Configura o código do status de resposta e envia a sua representação em sequência de caracteres como o corpo de resposta. |

Atividade



- Crie uma API calculadora.
- A API deverá ter:
 - Pelo menos 4 end-points do tipo GET.
 - Usar os arquivos externos com as 4 operações básicas.
 - O Front-End deve enviar para o servidor Node os números e o tipo de operação que será realizada.

